Final Year Project - Term 2 Report

EXPLORING EFFICIENT SIMILARITY SEARCH ALGORITHMS WITH K-NEAREST NEIGHBOR GRAPH

HU Evan | CSCI 4998 | 18 May, 2018

Table of Contents

Table of Contents 1

1 Introduction 3

1.1 Nearest Neighbor Search 3

1.2 Approximate Nearest Neighbor Search 4

2 Motivation 5

2.1 Increasing Complexity Challenge from Big Data 5

2.2 Wide Application of Similarity Search - Case Study 6

2.2.1 Application in record linkage and deduplication [13] 6

2.2.2 Application in search engines recommendation system [14] 7

2.2.3 Application in computer vision 9

2.3 Review of Recent Research Interest in K-Nearest Neighbor Graph Search 11

3 Overview of Common Techniques 12

3.1 Locality Sensitive Hashing 12

3.1.1 Notations 12

3.1.2 Algorithm 13

3.2 Principle Component Analysis Hashing 15

3.2.1 Notations 15

3.2.2 Algorithm 15

3.3 Graph Nearest Neighbor Search 17

3.3.1 Notations 17

3.3.2 Algorithm 17

4 Related Experiments and Results 19

5 Results and Analysis 20

5.1 KNN Graph Construction 21

5.2 Search on KNN graph with GNNS 22

5.3 Construct Approximate KNN Graph 25

5.4 Search on AKNN graph with GNNS 26

5.5 New GNNS with Hashing Based Seeding Method 28

5.6 PS-GNNS VS State-of-The-Art 30

6 References 31

1 Introduction

Given a query item q (or a set of query items), similarity search, also known as close item search or proximity search, is defined to search through an existing dataset, which usually consists of a significantly large volume of items with the same format as the query item, with the objective to find the item(s) that are most similar to the current query item. To decide how similar two items are, there are different ways to measure the similarity between a pair of items, including Euclidean distance, cosine similarity, Jaccard similarity, etc [29, 30].

In this project, we are trying to explore a graph search method and propose a novel algorithm design and implementation based on K-Nearest Neighbor (KNN) graph, where each vertex is connected to its nearest k neighbors. In the following two sections, we will give an overview of two categories of similarity search problems, namely the Nearest Neighbor search and its natural relaxation version, i.e., the Approximate Nearest Neighbor search, and show that the focus has been on the latter due to practical concerns.

1.1 Nearest Neighbor Search

The Nearest Neighbor (NN) search requires the item(s) of greatest similarity measure to the query item to be returned. NN search is a critical component in many learning algorithms such as clustering, retrieval and matching [1].

Once a certain similarity measure is adopted, a straightforward solution to NN search problem is to perform a linear search, exhaustively scanning through each item in the dataset (the "base" set, or simply the "base"), comparing them against the query and retrieving the one(s) with highest similarity. For a dataset with n items, each one of which is a d-dimension vector, time complexity of the linear search will be $O(nd)$. Despite the easiness of the implementation, it may not be

practical in reality. because the base set can contain millions of items and the items can range from 100 to 10,000 dimensions. Beyond the infeasibility of the computational cost for exhaustive search, the storage constraint originating from loading original data into memory also becomes a critical bottleneck [2].

There have been various tree-based structures developed to solve the NN search problem [3], [4]. However, the performance of these methods is even worse than a linear scan search when the dimensionality of the items is high [5]., not to mention that the tree-based structures require significant space complexity and sometimes takes up more storage than the original data itself Until now, few computationally feasible solutions have been proposed for this scenario.

1.2 Approximate Nearest Neighbor Search

Given the intrinsic computational difficulty of exact NN search problem, Approximate Nearest Neighbor (ANN) search has been receiving increasing interest in response. Instead of looking for the exact closest match, pseudo-optimal results are expected for ANN search. In fact, for many practical problems, such pseudo-closest results are shown to be enough and useful [6]. ANN search indeed has been commonly used not only in computer vision problems including image/video retrieval [7], recognition [8], and pose estimation [9], but also recommendation engines, anomaly detection and database linkage and deduplication

2 Motivation

In this section, we highlight the significance of studies of the similarity search problem and the incentive to explore efficient solutions for approximate search using the KNN graph.

2.1 Increasing Complexity Challenge from Big Data

There has been rapid growth in big data in the decade as the widely spread Internet has brought along a massive amount of information, thanks to the better connectivity and bandwidth brought by information technology advancement. Nowadays, the World Wide Web is expected to contain around 400 million accessible websites and more than 1 trillion webpages [2]. For instance, WhatsApp has a volume of around 55 billion messages per day and WeChat users send some 38 billion messages per day [10]. Twitter receives over 100 million tweets per day and Yahoo! exchanges over 3 billion messages per day. Besides the overwhelming textual data, the photo sharing website Flicker has more than 5 billion images available, where images are still kept being up loaded at the speed of over 3,000 images per minute. For the rich media sharing website YouTube, over 100 hours of videos are being uploaded per minute. [2].

The explosive increase in the volume of data has challenged the modern information technology in terms of not only maintaining a giant database but searching for relevant content in it. In fact, compared to the cost of storage, searching for relevant content in massive databases turns out to be even a more challenging task. In particular, searching for rich media data, such as audio, images, and videos, remains a major challenge since there exist major gaps between available solutions and practical needs in both accuracy and computational costs [2].

Besides the widely used text-based commercial search engines such as Google and Bing, content-based image retrieval (CBIR) has attracted substantial attention in the past decade [11]. Instead of relying on textual keywords based indexing structures, CBIR requires efficiently indexing media content in order to directly respond to visual queries. In these use cases, the linear search of $O(nd)$ time complexity is undesired given the realistic large-scale settings. Besides the scalability issue, most practical large-scale applications also suffer from the *curse of dimensionality* [12], when the data under modern analytics usually contains thousands or even tens of thousands of dimensions, e.g., in documents and images. Imagine a database containing 1 billion images and each one of them is represented as a vector in real coordinate space R^d. Suppose d is 1,000 (in realistic problems d is usually even larger), a naive query would need at least 1 trillion (1 billion * 1,000) times calculations of $O(1)$. For a server that can perform 10^11 operations per second (100 G), this would translate to 10 seconds of response time, which is unaffordable. One might argue that there could be multiple servers to distribute the computations, but on the other hand, there could be multiple queries from users at a time. In general, the expensive time complexity to exhaustively search through the base set with ever growing amount of data very much exceed the short response time expected by usual users.

2.2 Wide Application of Similarity Search - Case Study

The problem of similarity search is fundamental to many tasks, including database record linkage and deduplication, search engines and recommendation system and computer vision.

2.2.1 Application in record linkage and deduplication [13]

Record linkage is the process of matching records from several databases that refer to the same entities. When applied on a single database, this process is known as

deduplication, because that task will then be to remove duplicates of a record to prevent unnecessary storage usage. Record linkage can help collect information that is not available otherwise, or that is too costly to acquire; removing duplicate records is a crucial step in the data cleaning process, because duplicates not only result in extra storage cost but also severely affect subsequent data processing or data mining. Organizations such as government agencies, public organizations, businesses and research projects can run into the case when secondary information are collected instead of primary information acquisition which can sometimes be very costly. For instance, a retail banking initiative needs to collect credit card usage information for the population to decide on a marketing campaign. The marketing consultancy probably needs to gather information from credit card issuers where the data could have gone through data masking techniques to ensure privacy of the customers. To effectively detect the usage patterns, it is important to perform record linkage to enrich the eventual database. Because the data are masked and anonymous, there is no unique key across different databases to allow a simple *join* operation. Hence a similarity search can be adopted to cross-check for the records of the same entity.

2.2.2 Application in search engines recommendation system [14]

Web search engines like Google or Bing has become an integrated part of everyday Internet users nowadays. They provide a user-friendly interface that allows users to search by simply typing keywords related to their targets. Although it is easy for users to search the Web, a list of keywords do not always accurately describe what the user has in his/her mind . One reason for this is the intrinsic ambiguity that of natural languages. Queries having ambiguous terms may retrieve unexpected results for the user. On the other hand, users may phrase their queries very differently even when they are searching for the exact same piece of information. For instance, to search for the application of similarity search, one user might enter query as “realistic application of similarity search” while another user might

type in query like “how can similarity search be used in practice”. Sometimes the topic might involve certain technological terminologies about which the case: users may have little prior knowledge until they can obtain and study the right information; there are also some cases where users did not spell the keywords correctly.

In order to resolve these issues, many search engines have implemented recommendation systems to recommend potential alternatives that might lead the users to what they are actually after.

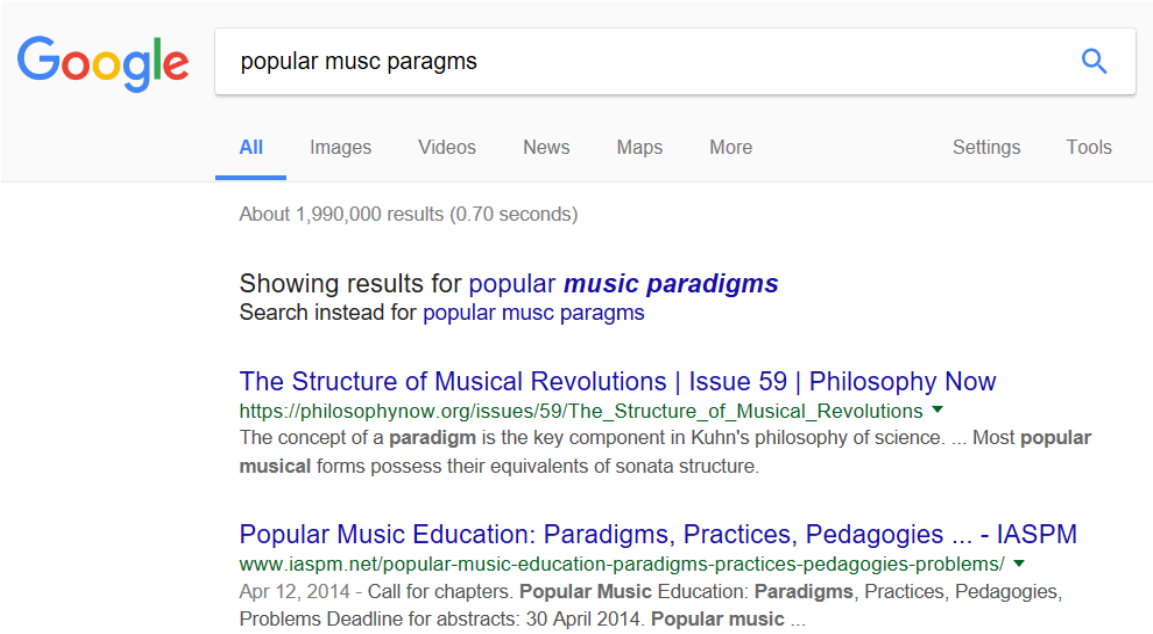

Figure 1: Google search engine suggesting alternative search keywords for users. The user mis-typed “popular music paradigms” as “popular musc paragms” and Google was able to make reasonable recommendation.

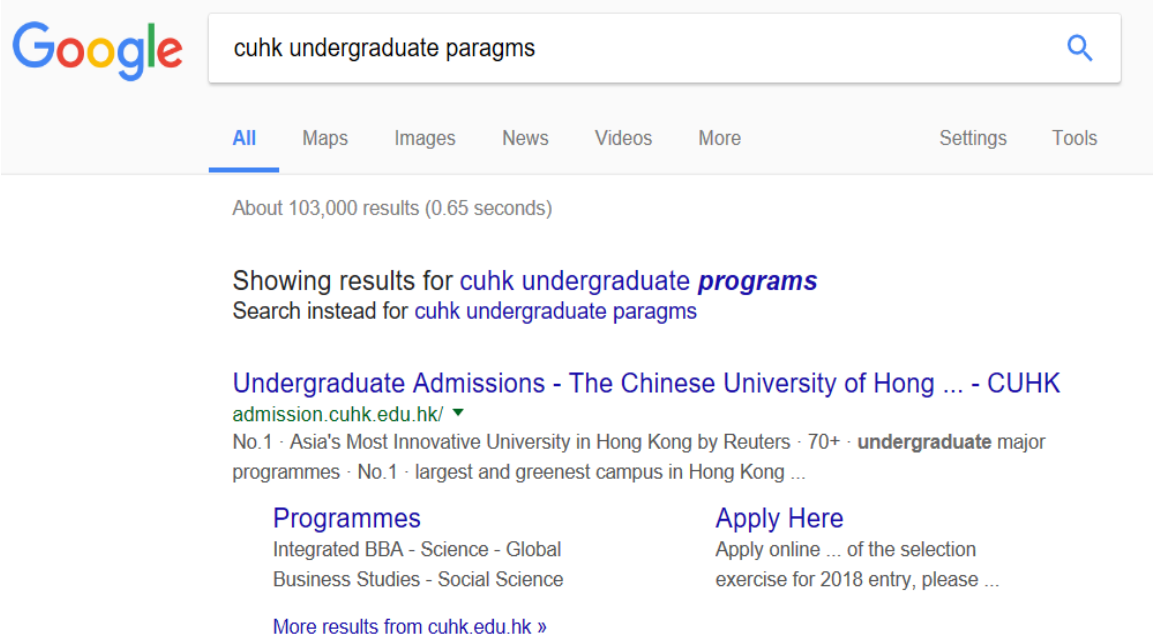

Figure 2: Google search engines identified and "understood" the context of the mistyped keyword "paragms" and suggested "programs" instead.

As shown in Figure 1 and 2, the mistyped keyword "paragms" could be matched to either "programs" or "paradigms", and the Google search engine is smart enough to "guess" the context of the queries and make efficient recommendations on the alternative search keywords for the users. The technique behind the screen actually involves similarity search, where potential queries are ranked by their similarity to the user's query, based on the URL's that were clicked for them. Similarity search plays a central role in the users' experiences with the search engine because people can easily feel if the engine is intelligently giving them what they want constantly.

2.2.3 Application in computer vision

Similarity search is widely used in computer vision tasks including image/video retrieval [7], recognition [8], classification [15] and pose estimation [9] For

instance, the police might have an image of a crime suspect, and also a database of photos of the citizens. In order to efficiently identify the closest matches, the techniques of similarity search of images can be deployed to assist the screening process. In this project, we will also explore novel algorithms design based on KNN graph using database of images.

Figure 3 [16]: Searching for local feature (nose and smile) matches of images. Same technique can be applied to search for matches of the entire image as a whole by including multiple local features.

In computer vision tasks, an image of $h * w$ definition can be represented by a vector in the real coordinate space of R^d, where each pixel translates to three real values (RGB values) hence $d = 3 * h * w$. To search for an image in the database that is similar to the query, the first method that would come into many people's mind may be using machine-learning based techniques like Support Vector Machine (SVM) or Convolutional Neural Network (CNN). However, there are advantages that similarity search methods have: (i) Avoid overfitting of parameters, because no parameter learning is required. (ii) Can naturally handle a huge number of classes. (iii) Require no training/learning phase [17] In the studies, a method named Naive-Bayes Nearest-Neighbor (NBNN) similarity search based methods are also be shown to perform in line with top leading learning-based image classifiers [17].

2.3 Review of Recent Research Interest in K-Nearest Neighbor Graph Search

There has been continuous interest in developing search methods based on KNN graph for the ANN search problem [18], [19] [20], [21], [22]. A KNN graph is a directed graph $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{E})$, where $\boldsymbol{X}$ is the vertex set containing the n items of d dimensions in the database ($\boldsymbol{X} \in \mathbb{R}^{d \times n}$) and $\boldsymbol{E}$ is the directed edge set in which vertex x_i is connected to vertex $x_j \Leftrightarrow x_j$ is one of the KNN of x_i. Paredes and Chvez [20] construct a KNN graph offline and utilize the graph constructed to use as few distance computations as possible for similarity measure during the online searching phase since the distance is considered expensive to compute. Paredes and Chvez present two search algorithms for both range and nearest neighbor queries which use navigational and metrical features of the KNN graph. It is shown that their approach is competitive against current ones. For instance, in the document metric space our nearest neighbor search algorithms perform 30% more distance evaluations than Approximating and Eliminating Search Algorithm (AESA) [23] using only a 0.25% of its space requirement. In the same space, the pivot-based technique is completely useless. Lifshits and Zhang [19] define a visibility graph for any dataset satisfying the disorder inequality, and adopt a greedy routing over the graph that can deterministically converge to the NN of a target in logarithmic number of steps. Hajebi et al. [18] also build a KNN graph in an offline phase and when queried with a new item, perform hill-climbing starting from a randomly sampled node of the graph. Theoretical guarantees for the accuracy and the computational complexity are provided and the effectiveness of their algorithm are also shown in [18]. Jin et al. [21] also construct a KNN graph in an offline phase and use a novel algorithm named Iterative Expanding Hashing (IEH), which builds an auxiliary index based on the KNN graph and expands multiple nodes at each iteration. This auxiliary index can be easily combined with all the traditional hashing methods to formulate the initial seeding, i.e.., choice of

initial nodes to be expanded. Zhao et al. [22] propose an efficient KNN graph construction method based on two means clustering, and adopt the enhanced hill-climbing similar to IEH but with the support of inverted indexing derived from residue vector quantization. We try to explore the KNN graph based search methods and, after studying these related works and familiarizing ourselves with relevant techniques, to propose novel KNN graph search methods with the objective to improve the performance of the promising search methodology.

3 Overview of Common Techniques

There are two major directions in the literature regarding the ANN search problem. One direction is hashing based, utilizing different vector quantization techniques to hash the items from a higher dimensional space to a lower dimensional space. Alternatively, a KNN graph is utilized to lead the search starting from a randomly picked (or computed) seed node to the query target. In this section, we give an overview of both paradigms and describe some common algorithms including Locality Sensitive Hashing, Principle Component Analysis Hashing and Graph Nearest Neighbor Search.

3.1 Locality Sensitive Hashing

The Locality Sensitive Hashing (LSH) algorithm is designed to covert an item to a binary vector of lower dimensions while preserving its locality metrics. The original version of LSH was introduced by Indyk and Motwani [24].

3.1.1 Notations

Let $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$ denote the base dataset with n items of d dimensions. ρ is the distance measure function and $\rho(x_i, x_j) = \| x_i - x_j \|_2$ is the l2-distance

between data point x_i and x_j. d_H is the Hamming distance function and $d_H(\boldsymbol{a}, \boldsymbol{b})$ is the Hamming distance between the two binary codes. To assign each data point x to a c-bit hash bucket, there are c hash functions to convert each data point to a c-bit hash code, which is essentially a c-dimensional binary vector

$$H(x) = [h_1(x), \dots, h_c(x)]$$

where $h_l(x) \in \{0, 1\}$ is the l-th hash function and $h_l(x) = sgn(xw_l^T)$ with w_l being a randomly generated weight vector.

3.1.2 Algorithm

The LSH algorithm consists of the following steps:

Input:

database $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$

number of bits c

search radius r_H (in terms of Hamming distance)

query item $\boldsymbol{q} \in \mathbb{R}^d$

top k results to be returned

Output:

k items $\boldsymbol{x} \in \boldsymbol{X}$ that are the approximate nearest neighbors of $\boldsymbol{q}$

Methodology:

I. Preprocessing: assign each data point in the database to a hash bucket
    a. Randomly generate l weight vectors from a zero-mean multivariate Gaussian $\mathcal{N}(0, \mathbf{I})$ of d dimensions
    b. For each $\boldsymbol{x}_i \in \boldsymbol{X}$, map it to the hash code $H(\boldsymbol{x}_i) = [h_1(\boldsymbol{x}_i), \dots, h_c(\boldsymbol{x}_i)]$

II. Searching: perform searching when a query $\boldsymbol{q}$ is raised

   a. Coding stage: map $\boldsymbol{q}$ using the same weight vectors generated in I.a. to get its hash code $H(\boldsymbol{q}) = [h_1(\boldsymbol{q}), \dots, h_c(\boldsymbol{q})]$
   b. Locating stage: collect all the database items in the hash buckets that fall within the Hamming distance radius r_H
   c. Scanning stage: perform a linear scan over the items collected and return the k items that are of minimum ρ measures with $\boldsymbol{q}$

Note that the Hamming distance can be cheap to compute using low-level hardware operation XOR.

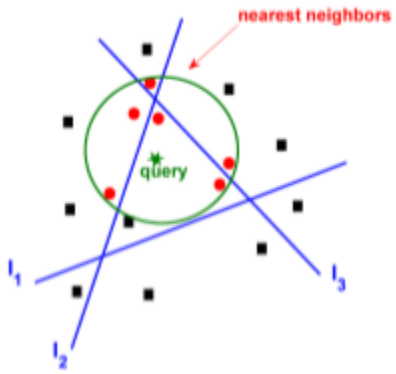

Figure 4 [21]: Example LSH in 2-D space. A hashing method generates three hyperplanes l1, l2, and l3 to separate the 2-D space into seven parts. The points in the same part have the same binary codes. The query point is the green star and its NN are represented by red circles. The remaining points are black squares.

As shown in Figure 4, the choice of the search Hamming radius r_H has a strong impact on the performance of the searching. In the above example, to achieve a 100% recall, one must specify the parameter to be $r_H \geq 2$. However, the trade-off exists because a larger r_H usually means more items in the database will be collected in the locating stage and hence more ρ distance computations in the scanning stage, which could be expensive given a large d.

3.2 Principle Component Analysis Hashing

Instead of generating hash functions randomly, the Principle Component Analysis Hashing (PCA Hashing, or PCAH) algorithm [30, 25]tries to "learn" the weight vectors in a scientific way based on the distribution of the items in the database. It uses a simple but efficient method to get the hash functions. It reduces dimension of the data utilizing the power of the PCA technique. Notations and steps are as following.

3.2.1 Notations

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ denote the base dataset with n items of d dimensions. ρ is the distance measure function and $\rho(\boldsymbol{x_i}, \boldsymbol{x_j}) = \| x_i - x_j \|_2$ is the l2-distance between data point $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. d_H is the Hamming distance function and $d_H(\boldsymbol{a}, \boldsymbol{b})$ is the Hamming distance between the two binary codes. To assign each data point x to a c-bit hash bucket, there are c hash functions to convert each data point to a c-bit hash code, which is essentially a c-dimensional binary vector

$$H(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \dots, h_c(\boldsymbol{x})]$$

where $h_l(x) \in \{0, 1\}$ is the l-th hash function and $h_l(\boldsymbol{x}) = sgn(\boldsymbol{x}\boldsymbol{w}_l^T - \mathbb{E}_k(\boldsymbol{x}_k \boldsymbol{w}_l^T))$ with w_l being a randomly generated weight vector.

3.2.2 Algorithm

The main motivation of the PCAH algorithm is to try to maximize the information contained in each bit of the resulting binary hash bucket codes. Each bit is essentially reflecting the projection on a principle component, a direction in the space of the original dimension that the items vary significantly. To maximize the information, PCAH tries to maximize the variance of each bit and also keep each bit uncorrelated to others, which is essentially to perform a PCA reduction before quantizing the values to binary codes.

Input:

database $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$

number of bits c

search radius r_H (in terms of Hamming distance)

query item $\boldsymbol{q} \in \mathbb{R}^d$

top k results to be returned

Output:

k items $\boldsymbol{x} \in \boldsymbol{X}$ that are the approximate nearest neighbors of $\boldsymbol{q}$

Methodology:

I. PCA: Perform PCA on the database $\boldsymbol{X}$ and get the l projection vectors as weight vectors for hashing

II. Assign each data point in the database to a hash bucket
    a. Generate weight vectors using the projection vectors
    b. For each $\boldsymbol{x}_i \in \boldsymbol{X}$, map it to the hash code $H(\boldsymbol{x}_i) = [h_1(\boldsymbol{x}_i), \dots, h_c(\boldsymbol{x}_i)]$

III. Searching: perform searching when a query $\boldsymbol{q}$ is raised
    a. Coding stage: map $\boldsymbol{q}$ using the same weight vectors generated in I.a. to get its hash code $H(\boldsymbol{q}) = [h_1(\boldsymbol{q}), \dots, h_c(\boldsymbol{q})]$
    b. Locating stage: collect all the database items in the hash buckets that fall within the Hamming distance radius r_H
    c. Scanning stage: perform a linear scan over the items collected and return the k items that are of minimum ρ measures with $\boldsymbol{q}$

3.3 Graph Nearest Neighbor Search

The Graph Nearest Neighbor Search (GNNS) algorithm [26] is a hill-climbing search method based on KNN graph. Although the algorithm takes an constructed KNN graph as input, the method to efficiently construct a KNN graph is also worth discussion. We will discuss this further in Section 5. In this section we focus ourselves on the GNNS algorithm itself.

3.3.1 Notations

Let $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$ denote the base dataset with n items of d dimensions. ρ is the distance measure function and $\rho(x_i, x_j) = \| x_i - x_j \|_2$ is the l2-distance between data point x_i and x_j. Let $\mathcal{N}_k(\boldsymbol{x})$ denote the set of k nearest nodes of data point $\boldsymbol{x}$. A KNN graph is a directed graph $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{E})$, where $\boldsymbol{E}$ is the directed edge set in which vertex $\boldsymbol{x}_i$ is connected to vertex $\boldsymbol{x}_j \Leftrightarrow \boldsymbol{x}_j \in \mathcal{N}_k(\boldsymbol{x}_i)$.

3.3.2 Algorithm

The GNNS algorithm takes input and involves two stages as following:

Input:

database $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$

KNN graph $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{E})$

number of restart seeds R

number of iterations T

number of expansions E in each iteration

query item $\boldsymbol{q} \in \mathbb{R}^d$

top k results to be returned

Output:

top k results to be returned

Methodology:

$\mathcal{S} \leftarrow \{\}$ is the set of visited nodes

$\mathcal{U} \leftarrow \{\}$ is the set of ρ distance measures of the visited nodes against $\boldsymbol{q}$

for $r = 1, \ldots, R$ **do**

Randomly select an item $\boldsymbol{y}_0$ from $\boldsymbol{X}$

for $t = 1, \ldots, T$ **do**

Update $\boldsymbol{y}_t = \ argmin_{\boldsymbol{y} \in \mathcal{N}_E(\boldsymbol{y}_{t-1})} \rho(\boldsymbol{y}, \boldsymbol{q})$

Update $\mathcal{S} = \mathcal{S} \cup \mathcal{N}_E(\boldsymbol{y_{t-1}})$

Update $\mathcal{U} = \mathcal{U} \cup \{\rho(\boldsymbol{y}, \boldsymbol{q}) \mid \boldsymbol{y} \in \mathcal{N}_E(\boldsymbol{y_{t-1}})\}$

Return the k items in $\mathcal{S}$ with minimum corresponding ρ in $\mathcal{U}$

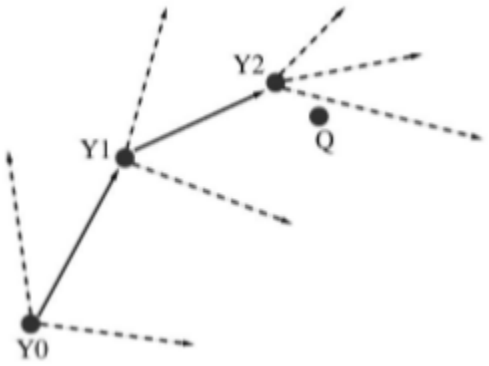

Figure 5 [26]: The GNNS algorithm on an example search path

As shown in Figure 5, starting from a randomly selected item, iterative hill-climbing takes place and lead the search to reach near the query target Q. It is not guaranteed that the true nearest neighbors will always be returned because the

search can be trapped in local optimum hence the purpose being serving ANN search problem instead of NN search one. To avoid being easily trapped in local optimum, R seeds will be selected in different trials. Obviously trade-off exists for R as well as T and E [26]: by increasing each of them, the algorithm spends more time in search and returns a more accurate result. The difference between E and k and K should be noted. E and K are two input parameters to the search algorithm (online), while k is a parameter of the kNN tree construction algorithm (offline).

4 Related Experiments and Results

To respond to users in time, the speed of a search algorithm is important. However in this context (ANN search), the time complexity alone cannot fully describe the performance of an algorithm, given that we are not solving NN search but instead ANN search problem. This implies that different algorithms differ not only in terms of time but also their ability to recall the ground truth, which is the set of the NN of the query. So the criteria to evaluate the performance of an algorithm should involve both time and search quality.

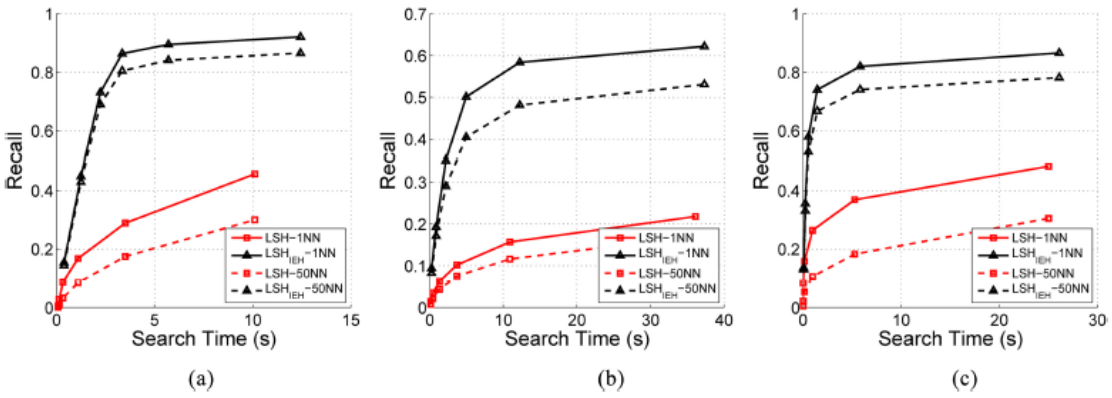

Figure 6 Recall quality of KNN VS LSH against time

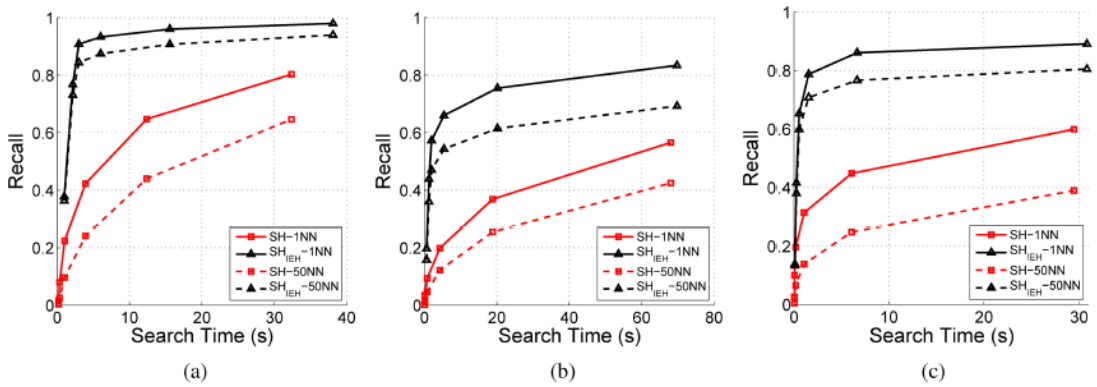

Figure 7 Recall quality of KNN VS SH against time

We try to replicate the above results done by Jin et al. [21], where the time-recall data is recorded for both hashing-alone and enhanced KNN (enhanced with hashing for initial seeds) on three different public datasets ((a) CIFAR-10. (b) GIST-1M (c). SIFT-1M), which are popular in the field. SH denotes for Spectral hashing which can be seen as a variation of PCAH and ITQ.

5 Results and Analysis

We first implement the construction of normal KNN. CIFAR60k is the dataset we first adopt, which contains 60,000 items of 512 dimensions. Gradually we improve the construction methods and use it on other datasets including GIST_10k and SIFT1m_10k. 5/10/20-NN graphs are constructed for each of the dataset. Note that GIST_10k is a dataset we build by sampling 10 thousand items from original GIST, and same for SIFT1m_10k.

GNNS is then applied on these constructed graphs to perform searching and the performance is recorded and compared against hashing-based algorithms such as PCAH and ITQ.

We then explore a novel approximate KNN (AKNN) method where we utilize PCAH during the construction of the KNN graph to get an approximate KNN graph. The recall of the approximate KNN on the exact KNN graph is computed to investigate the quality of the approximation.

Finally, we perform searching on the AKNN graph and compare its performance against that of KNN and hashing-based methods mentioned above.

5.1 KNN Graph Construction

We first set k=5 and target to build a 5-NN graph.

During the graph construction on dataset CIFAR60k, which contains 60,000 items of 512 dimensions, we notice that the naive construction method takes too long to construct the KNN graph. The naïve method (naïve_knn) constructs the graph in this way: For each item i, compute the l2-distance between item i and all other items j; Use quick-sort to sort all the other $n - 1 = 60000 - 1 = 59999$ items based on the l2-distance computed and retrieve $k = 5$ items that are closet to item i.

The method is inefficient because 1) it involves too many l2-distance computations. It performs $n \times (n - 1) \approx 3.5$ billion times of this similarity check, each one of which is of $O(d)\ where\ d = 512$ complexity; 2) quick-sort is actually more than we need and costs extra time, because we only need the top k results, instead of sorting all the results. In our experiment setting when the construction is run on proj99, it costs 20+ hours to finish. Therefore, we try to make some improvement.

We first notice that we can use a $n \times n$ matrix to store the distance. For example, when we compute distance between item 234 and item 567, we store the distance

to dist_map[234][567]. This way, when we reach $i = 567$ and $j = 234$, we can directly get the distance by referring to dist_map[j][i]. With this design, more memory complexity is incurred but half of the distance computations are reduced, which is desirable. Secondly, we adopt a findBestK() function instead of quick-sort to find the closet k neighbors. This function is of complexity O(N) (usually just 2n) which is better than O(NlogN) of quick-sort.

Originally it takes around 20 hours to construct the 5-NN graph and after the improvement only 10 hours are needed. The improved method (fast_knn) is used to construct the KNN graphs for k=5/10/20 on three datasets: CIFAR60k, GIST_10k and SIFT1m_10k.

Dataset	Cardinality	Dimension	Construction Method	Time	Graph Quality[1]
SIFT1m_10k	10,000	128	naïve_knn	4m46.990s	100%
SIFT1m_10k	10,000	128	fast_knn	2m51.005s	100%
SIFT1m_10k	10,000	128	pcah_aknn[2]	0m25.055s	84.17%

Table 1 KNN graph construction time and quality with different methods

Table 1 shows that how different construction methods perform. The details of the third method pcah_knn will be discussed in Section 5.3. Note that both naïve_knn and fast_knn are constructing exact KNN graph, hence their graph quality of 100%.

5.2 Search on KNN graph with GNNS

With the constructed graphs, we implement and apply the discussed GNNS search algorithm. A timer is also applied to control the time. We also compute the ground truth for each query using brute force manner with multi-threading, which allow us to compute the recall of the GNNS search until different time limits.

[1] Number of correct 20-NNs found
[2] To be discussed in Section 5.3

We then compare the time-recall performance of GNNS with hashing-based techniques including PCAH and ITQ. We test the results on K =5/10/20 NN graph and the results are following:

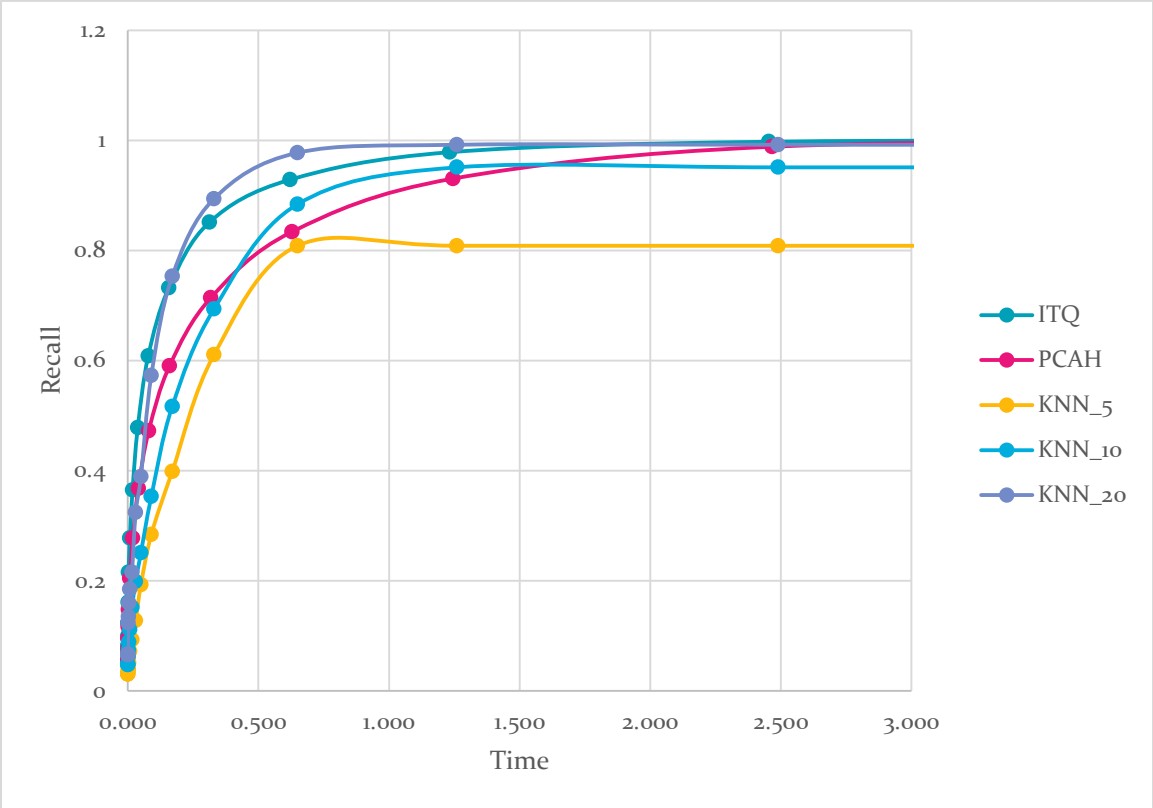

Figure 8 Time-recall search performance on CIFAR60k

We observe in Figure 8 that the 20-NN graph produces the best performance, beating the traditional hashing-based techniques including PCAH and ITQ.

In addition, the performance of GNNS enhances as K increases. That is, performance from KNN_5, KNN_10 to KNN_20 is gradually increasing. This makes intuitive sense because when k is larger, more items are available for probing and the search is less likely to be constrained in a small local community of nodes.

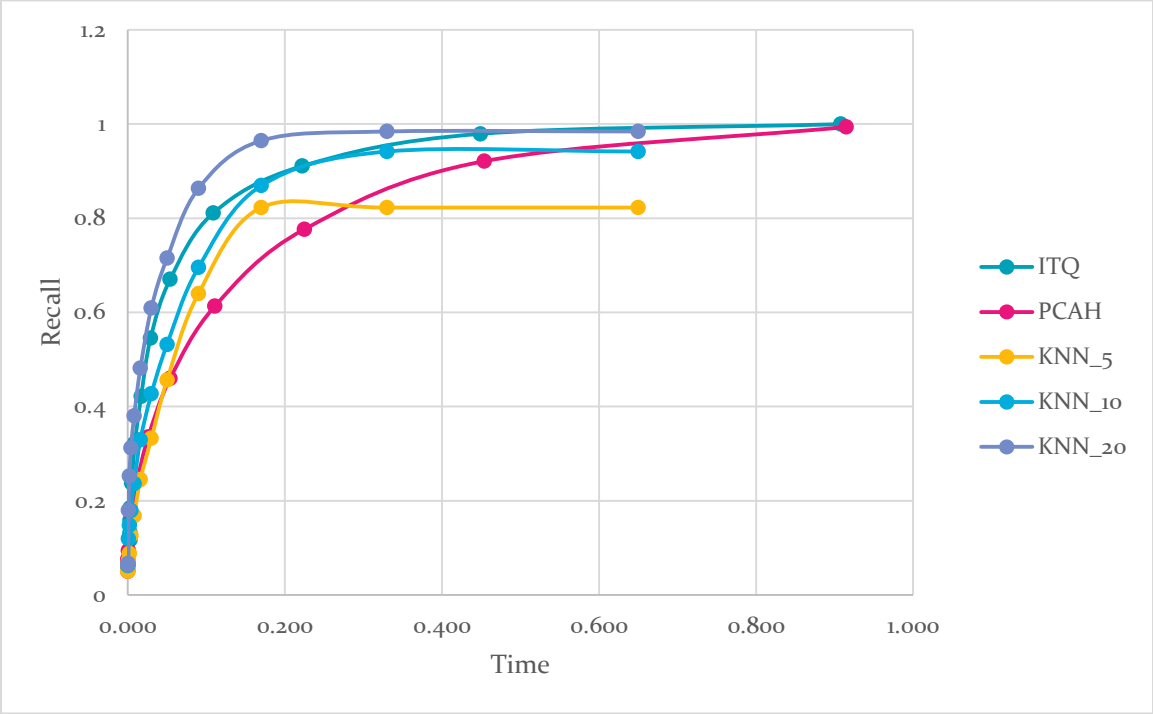

Figure 9 Time-recall search performance on GIST_10k

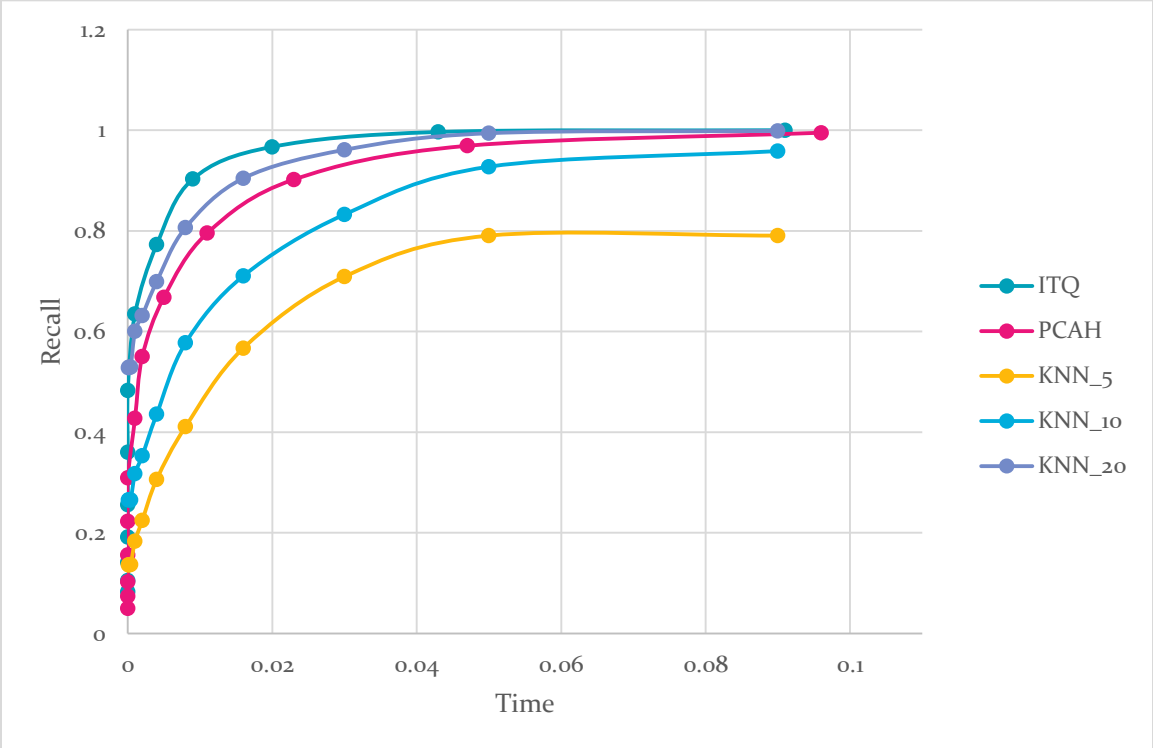

Figure 10 Time-recall search performance on SIFT1m_10k

Similar results are also observed on the other two datasets as in Figure 9 and 10. We do notice that the KNN_20 performance on SIFT1m_10k does not beat ITQ. We think it is because the parameter K is not large enough. With K increased, we should be able to see a boost in the performance.

The results suggest that our GNNS based on KNN can beat the traditional hashing based techniques including PCAH and ITQ, if K is appropriately selected.

5.3 Construct Approximate KNN Graph

We look to build a new KNN graph (named Approximate KNN graph, or AKNN graph) which can enable faster construction in offline phase and also update/query in online phase. The AKNN graph will be built such that each node is connected to k closest nodes that fall within buckets in certain radius of Hamming distance ($r_H, or\ maxHam$). In this design, we combine the traditional hashing techniques with the graph methods and hope to achieve less offline construction time while maintain a decent level of graph quality as well as search performance. Here we choose to use PCAH as the hashing technique to get the codebook.

To examine the quality of the approximation, we compute the graph quality, measured also by recall, of AKNN graph against respective KNN graph; i.e., a recall of 60% means the AKNN graphs on average captures 60% of true k nearest neighbors.

Code Len	8			Code Len	16			Code Len	32	
max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)
2	73.87	14.62		4	54.87	4.00		8	23.63	0.42
4	**98.55**	**63.76**		**8**	**99.02**	**59.81**		**16**	**99.01**	**56.87**

Table 2 AKNN graph quality - CIFAR60k

Code Len	8			Code Len	16			Code Len	32	

max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)
2	67.44	14.98		4	40.51	4.13		8	10.50	0.45
4	**97.46**	**63.65**		**8**	**97.39**	**59.79**		**16**	**96.22**	**56.93**

Table 3 AKNN qraph quality – GIST_10k

Code Len	8			Code Len	16			Code Len	32	
max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)		max Ham	Recall (%)	Scan Rate(%)
2	84.17	14.64		4	69.30	4.46		8	39.40	0.65
4	**99.37**	**62.36**		**8**	**99.67**	**59.01**		**16**	**99.79**	**56.24**

Table 4 AKNN graph quality – SIFT1m_10k

Table 2 and 3 show the results of AKNN graph construction. Scan rate denotes percentage of distance computations performed compared to fast_knn; i.e.,

$$Scan\ Rate = \frac{\#\ of\ dist\ computaion}{N \times (N-1)/2}$$

As shown in the table, the quality of the graph can be safely above 95% when the allowed Hamming distance difference is half of hashing code bit length. We will use the lower-right corner setting (CodeLen=32 and maxHam=16) for the below experiment in Section 5.4 and 5.5.

In experiments, we find that we can achieve roughly 85% recall when scan rate is 15% and 95% recall when scan rate is 55%. Therefore, with AKNN, the time required to construct or update the graph will be significantly lower, because much less items will be visited and computed distance against, nonetheless the quality of the graph is reserved at a good level.

5.4 Search on AKNN graph with GNNS

Again we perform GNNS but on the AKNN graph. Time-recall statistics are also recorded and compared against those of KNN and hashing-based methods. The

AKNN used here is an approximate 20-NN, based on 32-bit PCA hashing and 16-bit Hamming distance limit.

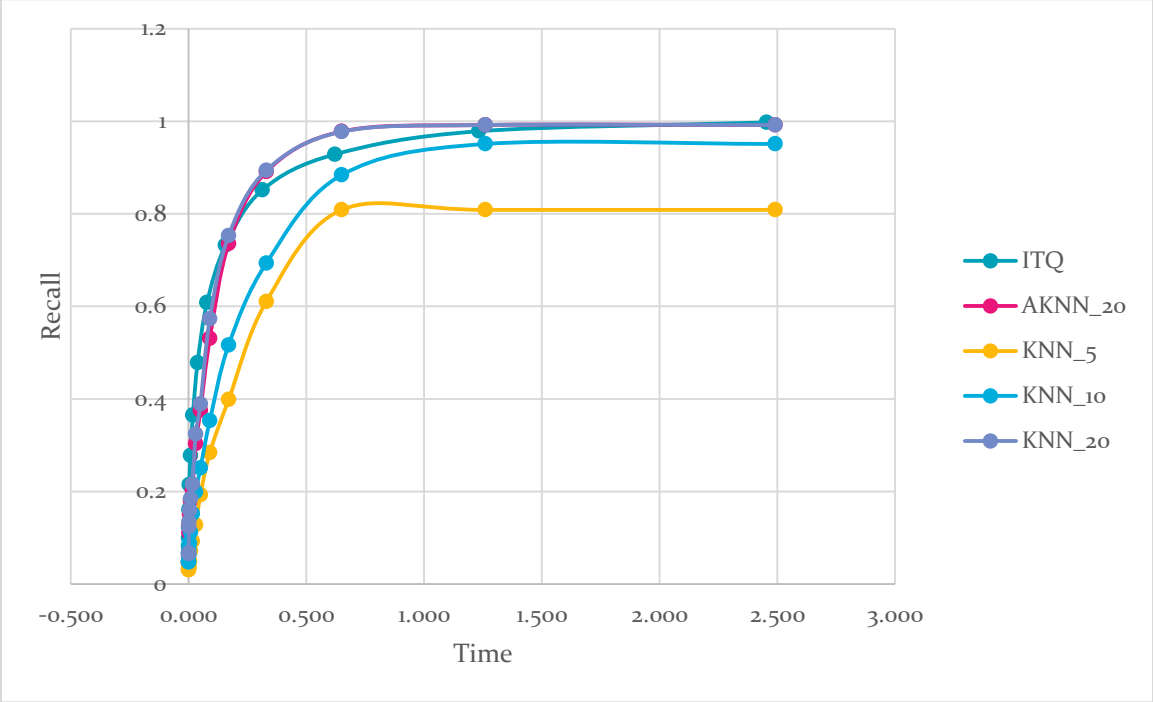

Figure 11 GNNS on AKNN_20 VS others – CIFAR60k

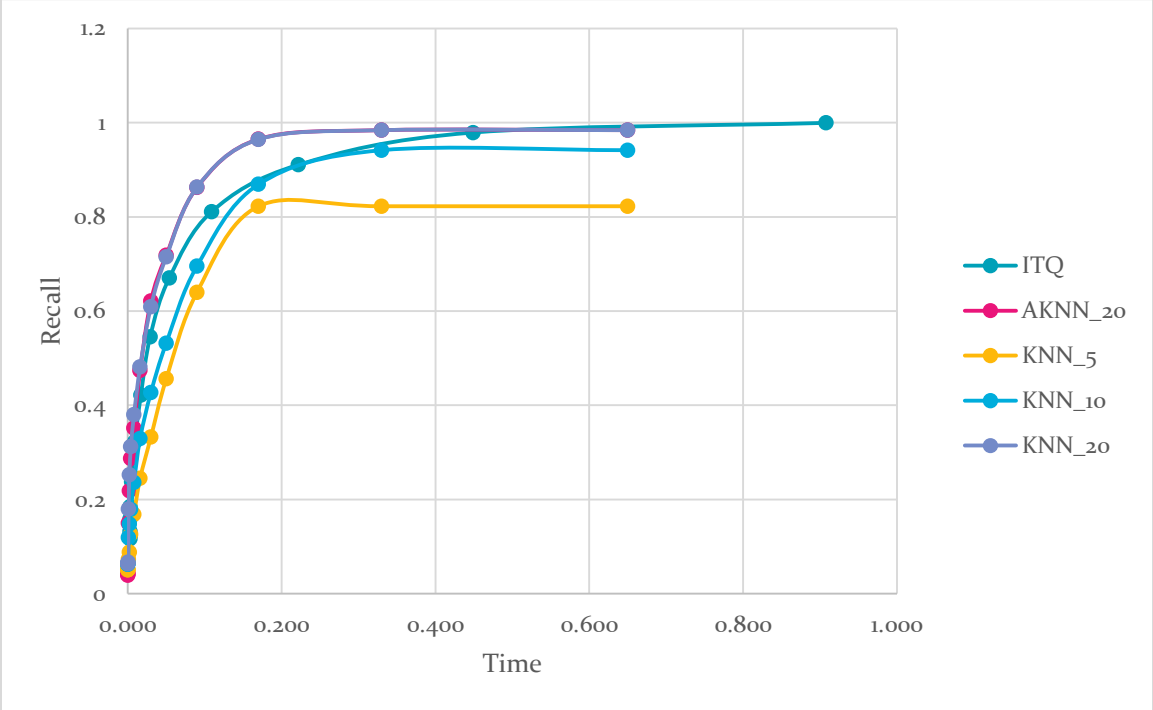

Figure 12 GNNS on AKNN_20 VS others – GIST_10k

The AKNN_20 almost coincides with KNN_20, suggesting that the two perform very much the same.

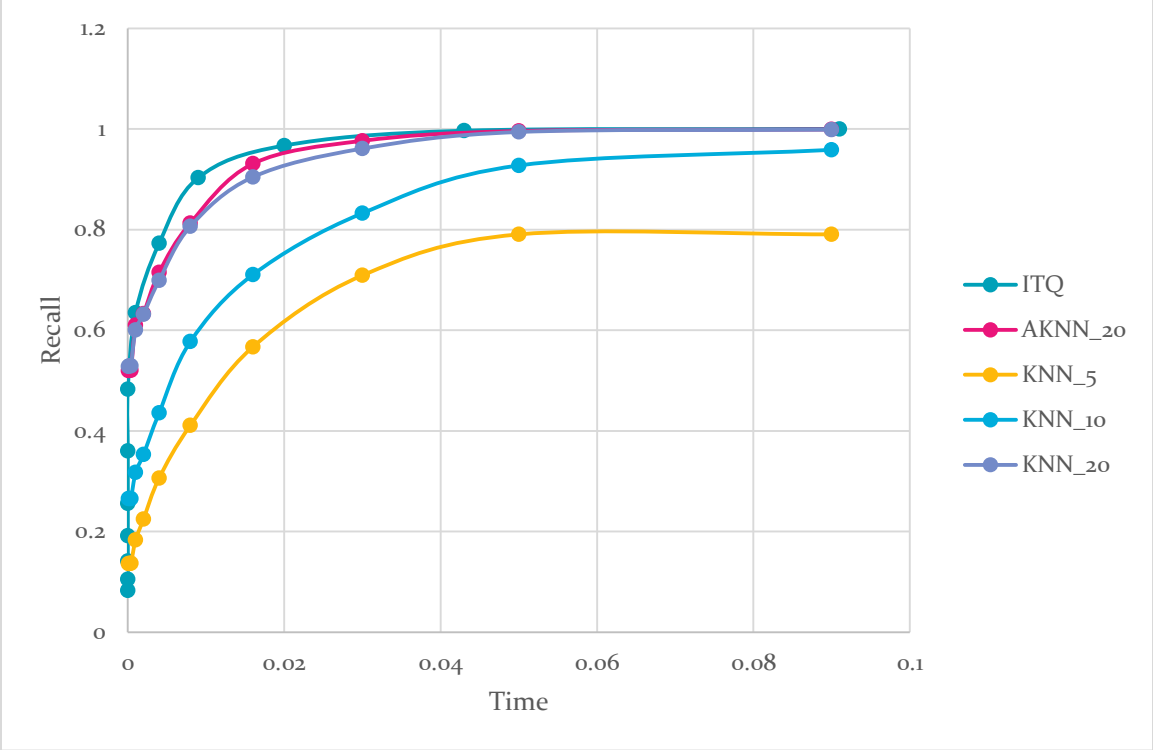

Figure 13 GNNS on AKNN_20 VS others – SIFT1m_10k

The results are very much consistent on SIFT1m_10k too. Note that as the time limits for the first few points are too small so the system may record the time differently on different runs even with same setting.

5.5 New GNNS with Hashing Based Seeding Method

We also design a new way to perform the searching, named as PCA Hashing-Seeded GNNS (PS-GNNS). Instead of randomly selecting items as seeds in GNNS, we utilize PCAH and compute the hash code of the query item. We then look in the dataset for the items that fall within a certain limit of Hamming distance with the query. Note that this process is also fast because the Hamming distance can be

computed at very low time expense by XOR operation. The items found will be used as seeds to perform normal GNNS.

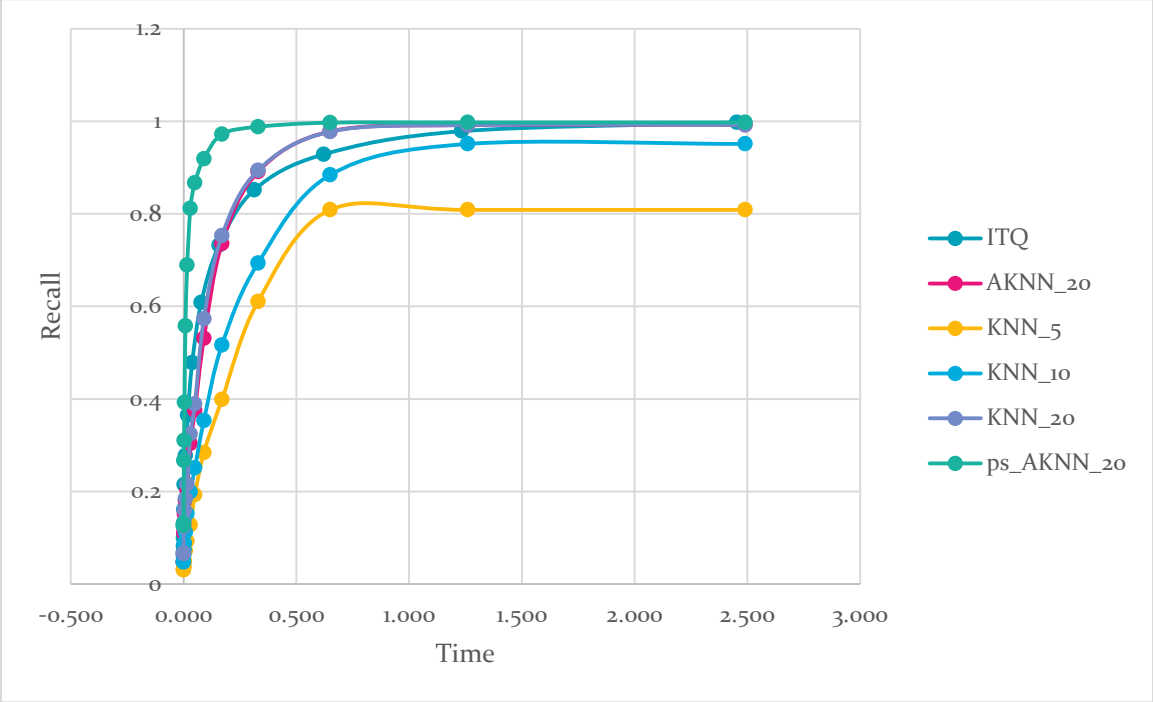

Figure 14 PS-GNNS results - CIFAR60k

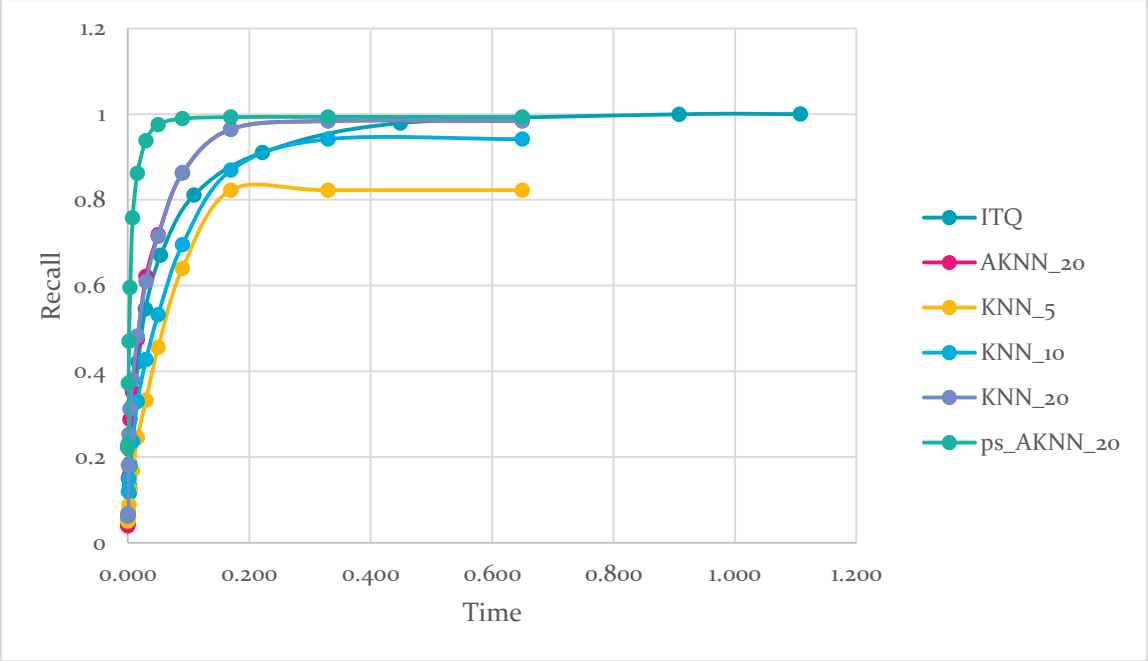

Figure 15 PS-GNNS results – GIST_10k

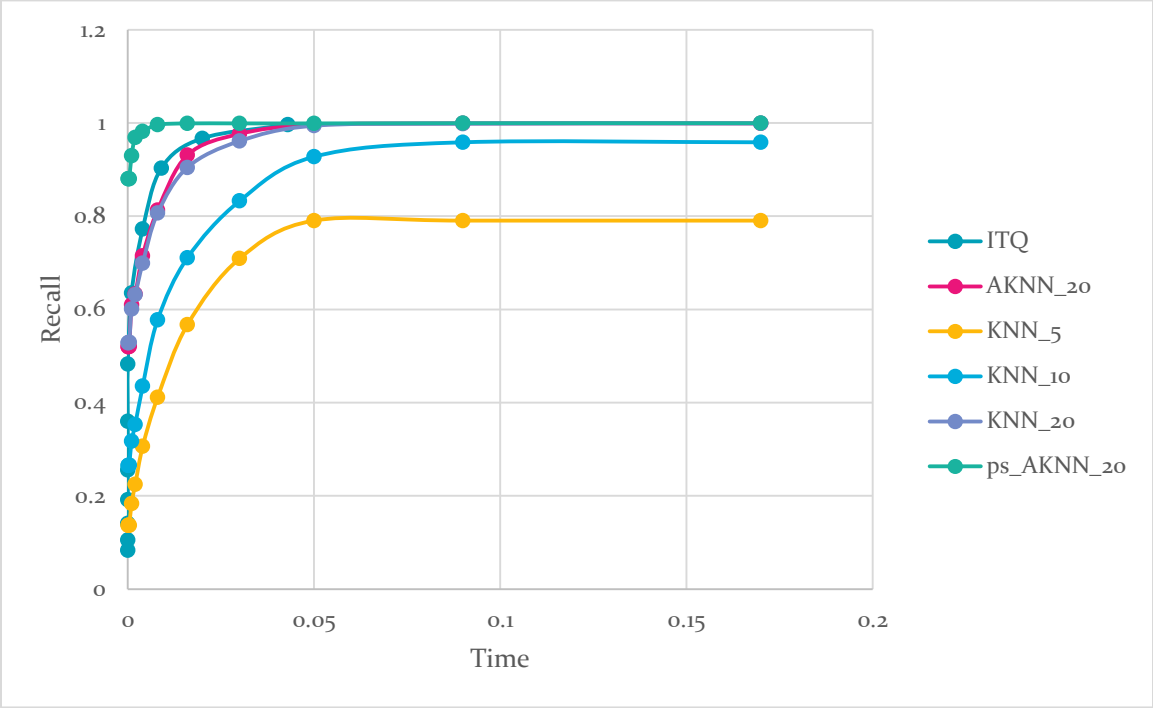

Figure 16 PS-GNNS results – SIFT1m_10k

The results suggest that this idea works very well. The PS-GNNS beats the others by quite a lot consistently across all three datasets. This shows that we can use PCAH to construct as well as improve searching on the AKNN graph.

5.6 PS-GNNS VS State-of-The-Art

Now we want to compare our best performing PS-GNNS result with the state-of-the-art methods. Before KNN methods were proposed, the state-of-the-art method should be Optimized Product Quantization (OPQ) [27]. Later, as pointed out in some studies [28], KNN methods tend to perform even better (not considering the eventual highest recall available). This suggests that with improvement brought by PCAH-based AKNN and novel searching technique PS-GNNS, our method will outperform the available algorithms in searching. Due to the difficulty of gathering their source code (or their source code is in Python, yet our apps are coded in C++), we do not carry out empirical experiment.

6 References

[1] J. Cheng, C. Leng, J. Wu, H. Cui and H. Lu, "Fast and Accurate Image Matching with Cascade Hashing for 3D Reconstruction," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[2] J. Wang, W. Liu, S. Kumar and S. Chang, "Learning to hash for indexing big data: A survery," in *IEEE*, 2015.

[3] L. Arge, M. Berg, H. Haverkort and K. Yi, "The priority R-tree: A practically efficient and worst-case optimal R-tree," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.

[4] J. Freidman, J. Bently and R. Finkel, "An algorithm for finding best matches in logarithmic expected time," in *ACM Trans. Math. Softw*, 1997.

[5] K. Beyer, J. Goldstain, R. Ramakrishnan and U. Shaft, "When is 'nearest neighbor' meaningful?," in *Proc. Int. Conf. Database Theory*, 1999.

[6] J. Wang, H. Shen, J. Song and J. Ji, "Hashing for similarity search: A survey," in *Uni. Princeton* , 2014.

[7] J. Sivic and A. Zisserman, "Video google: a text retrieval approach to object matching in videos," in *ICCV*, 2003.

[8] A. B. Torralba, R. Fergus and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," in *TPAMI*, 2008.

[9] G. Shakhnarovich, T. Darrell and P. Indyk, "Nearest neighbor methods in learning and vision: theory and practice," in *The MIT Press*, 2006.

[10] ASEAN, "The Star," Star Media Group Berhad, 12 Nov 2017. [Online]. Available: https://www.thestar.com.my/news/regional/2017/11/12/wechat-users-send-38-billion-messages-daily/. [Accessed 28 12 2017].

[11] R. Datta, D. Joshi, J. Li and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," in *ACM Computing Surveys*, 2008.

[12] R. Bellman, "Dynamic programming," in *1957*, Uni. Princeton.

[13] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," in *IEEE Trans. Knowledge Data Eng.*, 2011.

[14] R. Baeza-Yates, C. Hurtado and M. Mendoza, "Query recommendation using query logs in search engines," in *Uni. Valparaiso*, 2004.

[15] O. Boiman, E. Shechtman and M. Irani, "In defense of nearest-neighbor basedimageclassification," in *CVPR*, 2008.

[16] W. J. Scheirer, N. Kumar, P. Belhumeur and T. E. Boul, "Multi-attribute spaces: Calibration for attribute fusion and similarity search," in *CVPR*, 2012.

[17] O. Boiman, E. Shechtman and M. Irani, "In defense of nearest-neighbor basedimageclassification," in *CVPR*, 2008.

[18] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proc. 22nd Int.*

Joint Conf. Artif. Intell., 2011.

[19] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *ACM-SIAM Symp. Disrcrete Algo.*, 2009.

[20] R. Paredes and E. Chvez, "Using the k-nearest neighbor graph for proximity searching in metric spaces," in *Proc. 12th Int. Conf. String Process. Inform. Retrieval*, 2005.

[21] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, M. IEEE and X. He, "Fast and accurate hashing via iterative nearest neighbors expansion," in *IEEE Transactions on Cybernetics*, 2014.

[22] W. Zhao, J. Yang and C. Deng, "Scalable nearest neighbor based on knn graph," in *CVPR*, 2017.

[23] E. Vida, "An algorithm for finding nearest neighbors in (approximately) constant average time," in *Pattern Recognition Letters*, 1986.

[24] R. Motwani and P. Indyk, "Approximate nearest neighbor - Towards removing the curse of dimensionality," in *Proc. 30th Symp. Theory of Computing*, 1998.

[25] B. Wang, Z. Li and M. Li, "Efficient duplicate image detection algorithm for web images and large-scale database," in *Micorsoft Research*, 2005.

[26] Z. Jin, D. Zhang, Y. Hu, S. Lin and D. Cai, "Fast and accurate hashing via iterative nearest neighbor expansion," in *IEEE Trans. Cybernetics*, 2014.

[27] T. Ge, K. He, Q. Ke and J. Sun, "Optimized Product Quantization for Approximate Nearest Neighbor Search," in *CVPR*, 2013.

[28] D. Cai, "A Revisit of Hashing Algorithms for Approximate Nearest Neighbor Search," in *ACM*, 2018.

[29] J. Li, J. Cheng, F. Yang, Y. Huang, Y. Zhao, X. Yan, and R. Zhao, "LoSHa: A General Framework for Scalable Locality Sensitive Hashing," in *SIGIR*, 2017.

[30] J. Li, X. Yan, J. Zhang, A. Xu, J. Cheng, J. Liu, K. W. Ng, T. Cheng:, "A General and Efficient Querying Method for Learning to Hash," in *SIGMOD*, 2018.